PROGRAMACION: INTRODUCCION Y ALGORITMOS

Departament de Matemàtica Aplicada
i Telemàtica

Secció
Enginyeria
Telemàtica

INDICE

1.	**INTRODUCCION**	1
1.1.	La información	1
1.2.	Tratamiento de la Información	2
1.3.	Fundamentos del ordenador	5
	- Características básicas	5
	- Bloques principales	6
	Codificacion y Entrada/Salida	6
	Programa y lenguaje de programación	6
	Memoria Central y Memoria Secundaria	7
	Unidad Central de Proceso	9
1.4.	Algoritmo y Programa	11
	- Planteamiento	11
	- Análisis	11
	- Codificación del programa fuente	12
1.5	Ficheros y Sistema Operativo	13
	- Ficheros	13
	- Sistema Operativo	13
	- Editores y Compiladores	14
1.6.	Proceso de Edición, Compilación y Ejecución de programas	15
	- Proceso de edición, compilación, montaje y ejecución en Vax-Pascal	17
	- Compilación mediante colas de Batch	18

2. ALGORITMOS 20

2.1. Representación. Diagramas de Flujo 20
- Simbología ANSI de diagramas de flujo 21

2.2. Variables. Tipos de datos 22
- Lectura 23
- Escritura 23
- Expresiones 25

2.3. Estructuras algorítmicas básicas 27
- Estructura secuencial. Sentencia compuesta 28
- Estructura alternativa. Sentencia IF 29
- Estructura repetitiva. 32
  - Sentencia WHILE 32
  - Sentencia REPEAT 33

2.4. Traza de algoritmos. Ejemplos 35
- Ejemplos de algoritmos. Codificación en Pascal 35
  - Ejemplo 1. Máximo común divisor 35
  - Ejemplo 2. Factorial 38
- Traza de algoritmos o programas. Depuración 40

1. INTRODUCCION

1.1. LA INFORMACION

Definimos la "Información", en general, como todo aquello que hace incrementar nuestro nivel de conocimientos.
Las personas perciben a través de los sentidos todo tipo informaciones que provienen del mundo exterior. Sin embargo, dada la gran cantidad de informaciones que se pueden percibir, solamente consideraremos aquellas que realmente influyan en sus decisiones o comportamiento futuro.

Para la existencia de la información normalmente habrán de existir los siguientes elementos:

- Emisor de la información (Fuente).
- Medio que permite la comunicación de la información (Canal).
- Receptor que pueda comprender la información que recibe.

Para que el receptor pueda entender con facilidad la información recibida, y para que ésta pueda contener un significado más elaborado o complejo, se utilizan los "mensajes", los cuales hacen uso de un lenguaje conocido por el emisor y el receptor. Las formas de uso del lenguaje más importantes son:

- Lenguaje hablado (comunicación oral)
- Lenguaje escrito (comunicación visual).

Los mensajes empleados en la comunicación oral, en su contenido fundamental, pueden ser descritos también mediante mensajes del lenguaje escrito. Nos referiremos únicamente a éstos últimos.
Los mensajes del lenguaje escrito hacen uso de un número finito de símbolos (letras, números, signos) que constituyen el alfabeto, así como de unas reglas o normas propias del lenguaje empleado (ortografía, gramática, etc.)

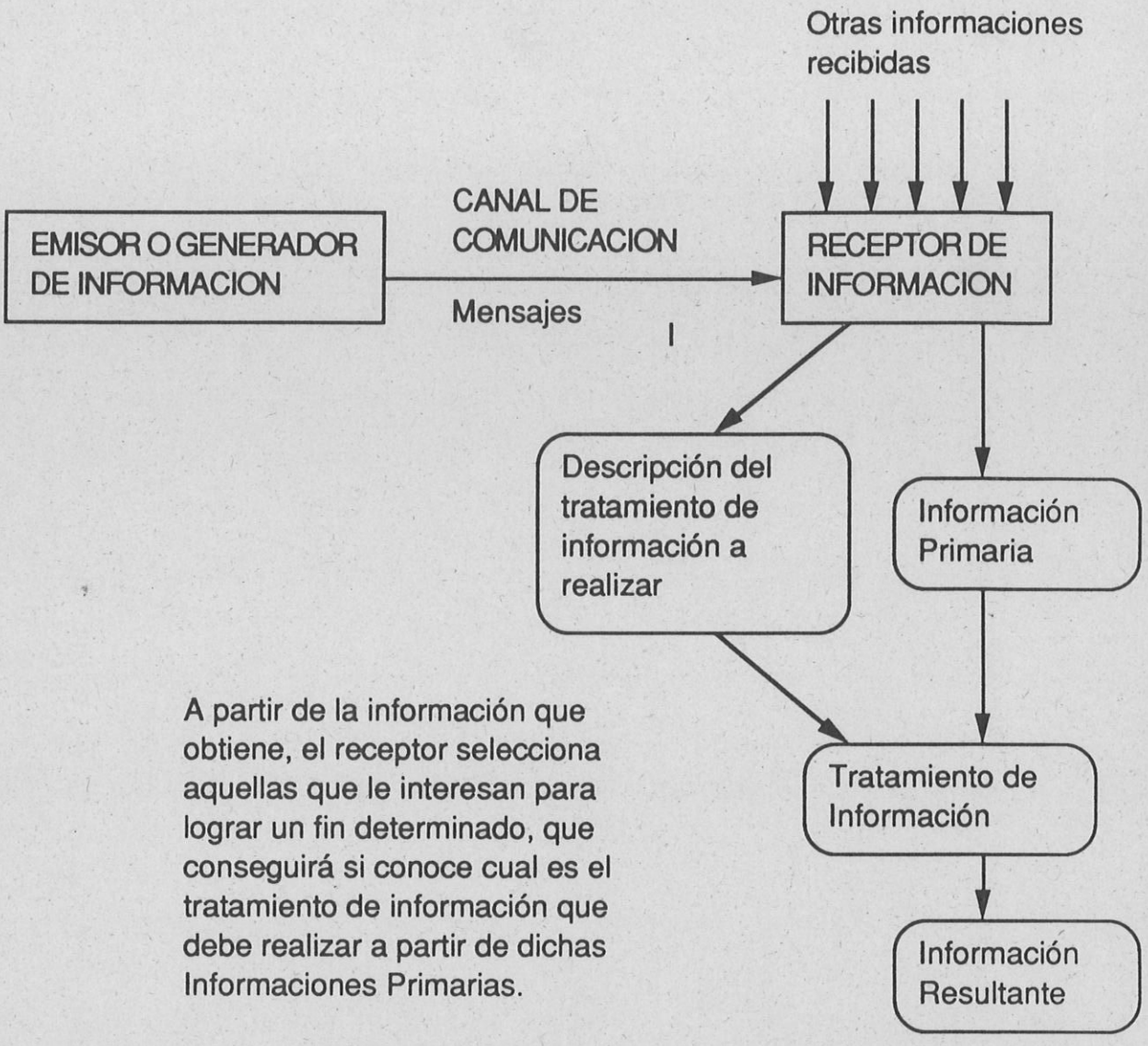
De las informaciones que percibe el receptor sólo consideraremos aquellas que éste va a emplear para conseguir un determinado proposito o fin, y que llamaremos Información Primaria.
Supondremos conocidos también qué es lo que se desea realizar con dicha información primaria, o sea, el proceso o tratamiento que habrá de aplicarse para llegar al fin deseado, el cual podrá consistir, por ejemplo, en la obtención de cierta información resultante de dicho proceso.

1.2. TRATAMIENTO DE LA INFORMACION

Llamaremos **Tratamiento de la información** a todo el conjunto de procesos u operaciones en general que habrán de realizarse partiendo de las informaciones primarias para conseguir una finalidad previamente determinada, obteniendo así un conjunto de informaciones resultantes deseadas.

El proceso a realizar puede ser de naturaleza muy diversa. Por ejemplo: La clasificación, manipulación, ordenación o cálculos estadísticos a partir de un gran conjunto de "datos". O también, a partir del modelo matemático que rige el comportamiento de un sistema físico, obtener la respuesta de dicho sistema frente a diversas situaciones que puedan darse en la realidad, o su evolución a partir de situaciones dadas, de acuerdo con las limitaciones asumidas en el modelo.
Las tareas que describen al tratamiento de información a realizar pueden contemplar la toma de decisiones entre varias alternativas en determinados momentos. En el caso más simple, en determinados puntos del proceso, se evaluarán ciertas condiciones, y según sean éstas ciertas o falsas, el proceso continuará por un camino u otro.
En cualquier caso, si se desea llegar con éxito al fin deseado, habrán de conocerse sin ninguna ambigüedad las tareas que deberán realizarse en todas las etapas y caminos involucrados en el proceso.

Resumiendo lo expuesto, el tratamiento de información lo podríamos representar en el siguiente esquema:



A partir de la información que obtiene, el receptor selecciona aquellas que le interesan para lograr un fin determinado, que conseguirá si conoce cual es el tratamiento de información que debe realizar a partir de dichas Informaciones Primarias.

En multitud de casos nos encontramos con tratamientos de información de naturaleza tal que, empleando los medios convencionales, se hace necesaria la participación de un gran número de personas o bien una gran cantidad de tiempo, o ambos a la vez, para lograr el fin deseado. También puede ser necesario realizar un mismo tratamiento para un gran número de casos distintos de informaciones primarias.

Es en estos casos donde se hace necesaria la colaboración de máquinas que permitan realizar la totalidad o la mayor parte del tratamiento de forma automática, para lograr así el ahorro de tiempo y de medios.

Definimos el **Tratamiento Automático de la Información** como aquel tratamiento de información en el cual las tareas descritas en el mismo se realizan de forma automática, mediante máquinas, sin la intervención humana. Sólo puede haber intervención humana para aportar la información primaria y la descripción de tareas a realizar, y en la interpretación de la información resultante del tratamiento.
El **Ordenador**, o Computador, es la máquina que permite la realización del tratamiento automático.
Con la necesidad de conocer el funcionamiento y utilización de los ordenadores aparece la **Informática**, que es la ciencia que estudia aquellos los temas relacionados con el tratamiento automático de la información.

Como en todo tratamiento de información, en el tratamiento automático deberán existir los siguientes elementos:

- Recursos físicos: Ordenadores.
- Informaciones primarias.
- Descripcción de las tareas a realizar en todas las etapas del proceso, para llegar al fín deseado.
- Informaciones resultantes.

El ordenador será así una máquina de uso general, de tal manera que aportándole de alguna forma la información primaria y la descripción de las tareas que debe realizar con la misma, nos habrá de suministrar la información resultante deseada, para cualquiera de los casos que puedan darse en dichas informaciones primarias. Y aprotándole otra descripción de tareas, la misma máquina habrá de servir también para otro tratamiento de información distinto.

1.3. FUNDAMENTOS DEL ORDENADOR

Características básicas.

En primer lugar, consideraremos las características en que se basa el funcionamiento de cualquier ordenador:

1. **Información binaria** Debido a la naturaleza de los dispositivos electrónicos y magnéticos en que se fundamenta, capaces de distinguir con facilidad entre dos niveles de una magnitud física, el ordenador puede almacenar mensajes empleando un alfabeto de sólo dos símbolos, que identificaremos como **0** y **1**, a diferencia de los alfabetos empleados en nuestros mensajes, que utilizan un número de símbolos muy superior. Cada uno de los símbolos 0, 1, (símbolos binarios) se denomina **bit**
La información que utiliza el ordenador estará pues representada en mensajes binarios (secuencias de bits).

2. **Almacenamiento de información**. El ordenador tiene la capacidad de "almacenar" de forma ordenada un elevado número de bits, normalmente agrupados en bloques de 8, 16 o 32 bits. Por ello, para referirnos a cierta cantidad de información binaria, se emplean también otras unidades derivadas de ésta información mínima (bit):
Un grupo de 8 bits, se denomina **byte**.
Un grupo de 2^10 bytes = 1024 bytes, se denomina Kilobyte, **Kb**, K.
Un grupo de 2^10 Kb = 1024 Kb, se denomina Megabyte, **Mb**, M.
Un grupo de 2^10 Mb = 1024 Mb, se denomina Gigabyte, **Gb**, G.
Normalmente, un ordenador podrá almacenar una cantidad de información binaria del orden de algunos Mb, con la capacidad de realizar la "lectura", "escritura" o modificación de dicha información con una enorme rapidez, (en tiempos del orden de los nanosegundos.)

3. **Realización de operaciones** El ordenador es capaz de realizar únicamente operaciones o transformaciones muy simples con la información binaria que tiene almacenada. Estas operaciones o "instrucciones elementales", que habrán de estar almacenadas en binario, las puede realizar también con una gran rapidez, del orden de los nano ó microsegundos.

Bloques principales.

Analicemos ahora cuales habrán de ser las funciones y los **elementos constituyentes básicos** de cualquier ordenador, teniendo en cuenta las características básicas expuestas anteriormente.

1. Codificación y Entrada/Salida. Debemos aportar al ordenador la información primaria y la descripción de las operaciones o tareas a realizar a partir de la misma. El ordenador nos habrá de presentar la información resultante. Estas informaciones deben ser fáciles de manejar para el ser humano, por lo que la forma básica de representación será el uso de mensajes escritos empleando nuestro alfabeto. Dado que el alfabeto empleado por el ordenador es el binario, esto nos lleva a la necesidad de que existan dispositivos que permitan la entrada y salida de mensajes en nuestro alfabeto, así como la conversión de información entre ambos alfabetos, el nuestro y el de la máquina (binario).
De esta función suelen encargarse varios dispositivos que denominamos **Periféricos**, (teclado, monitor, impersora, etc.) que permiten y facilitan la comunicación entre el usuario y la máquina.
Así pués, estos dispositivos realizarán el proceso de tradución de mensajes de nuestro alfabeto a mensajes binarios, denominado **codificación**, o el proceso inverso, denominado **decodificación**.
La intercomunicación entre estos dispositivos y el resto de bloques del ordenador, canalizando adecualamente el flujo de información, la habrá de realizar otro dispositivo, al que denominaremos **Unidad de Entrada/Salida.**

2. Programa y Lenguaje de Programación. La descripción del tratamiento de información que habrá de realizar el ordenador utilizará mensajes de nuestro alfabeto que describan dichas tareas. El ordenador estará diseñado con el fin de interpretar correctamente las tareas que debe realizar a partir de dichos mensajes. Por tanto, el lenguaje empleado en dichos mensajes habrá de seguir ciertas reglas establecidas de antemano. Además, será muy útil que en diferentes máquinas pueda ser empleado el mismo "lenguaje" en la descripción de tareas. Simplificando el proceso, la descripción de cada una de las tareas a realizar la denominaremos **"sentencia"**, y el conjunto de sentencias que permiten realizar un determinado tratamiento de información lo denominaremos **"programa"**. El lenguaje empleado en la descripción de las sentencias de un programa lo denominamos **"lenguaje de programación"**. Existen diversos lenguajes de

programación, cada uno con sus propias reglas de uso. Estas reglas se pueden clasificar en dos grupos:

- **Reglas sintácticas**: Indican la forma correcta de escribir individualmente sentencias válidas del lenguaje.
- **Reglas semánticas**: Indican la forma correcta de utilizar las sentencias del lenguaje, teniendo en cuenta su significado y su relación con el resto de sentencias.

En general, la complejidad de las sentencias de los lenguajes es superior a la complejidad de las operaciones o "instrucciones elementales" que el ordenador es capaz de ejecutar, por lo que cada sentencia del programa habrá de ser "traducida" por el ordenador en una serie de instrucciones más sencillas, que éste puede "ejecutar" directamente, y que habrán de representarse en el alfabeto binario. Diremos que el programa está **"en ejecución"** cuando el ordenador está realizando las instrucciones elementales que se han obtenido a partir de dicho programa. Tenemos así el **Programa Fuente,** escrito en un Lenguaje de Programación fácil de utilizar por el ser humano, y el **Programa Ejecutable**, que escrito en **Lenguaje Máquina** (codificado en binario) corresponde al programa que realmente se ejecuta en la máquina, y que es el resultado de la traducción del programa fuente en una
serie de "instrucciones elementales" que el ordenador puede realmente realizar.

3. Memoria Central y Memoria Secundaria. Para que el programa pueda ser ejecutado de forma automática, de manera que no deba introducirse en el ordenador cada sentencia en el momento en que deba ejecutarse, el programa deberá estar registrado o "almacenado" de alguna forma en el ordenador. De igual forma, las informaciones primarias, así como las que se vayan obteniendo durante el proceso, también habrán de estar almacenadas para que el programa pueda utilizar dichas informaciones cada vez que lo requiera. El dispositivo encargado de almacenar los programas y las informaciones necesarias para su funcionamiento se denomina **Memoria Central**.

Debe recordarse que la memoria central únicamente contiene información binaria, por lo que se habrá realizado previamente un proceso de codificación de la información (programas y datos). Para facilitar la localización de la información binaria en la memoria principal, ésta se organiza en una secuencia de "bloques" de bits, denominados **palabras**, cuya longitud depende del ordenador empleado, y pueden ser de 8, 16 o 32 bits, normalmente. Cada palabra, según su posición en la memoria, tiene asociada una **dirección**, (también codificada en binario, con un número

determinado de bits, que posibilitará la localización de cada "palabra" de la memoria). De esta manera, el contenido de la memoria puede expresarse como una secuencia de palabras ordenadas por su dirección (mapa de memoria). Cada "instrucción elemental" del programa (que puede constar de una o varias palabras), o cada dato que utiliza el programa (también de una o varias palabras) quedan así localizados en la memoria. La memoria principal debe permitir procesos de **lectura**, obteniendo la información que contiene, y de **escritura**, aportando información a la misma o modificando la que previamente contenía. Para que éstos procesos sean lo suficientemente rápidos para la ejecución de programas, la tecnología actual utiliza memorias de capacidad de almacenamiento limitada (centenares de Kb o algunos Mb), debido también a su relativamente elevado coste. Además, para su funcionamiento, precisa (en su mayor parte) de un aporte de energía externa. Ello provoca que la información que contiene, basada en dos estados de una magnitud eléctrica, no sea permanente, ya que al desaparecer esta alimentación de energía se pierde su contenido.

En relidad, para que ordenador pueda ser utilizado, deben existir unos programas que permitan la correcta puesta en marcha y la realización de operaciones fundamentales con sus dispositivos. Por razones de seguridad, estos programas estarán en una zona de memoria cuyo contenido será permanente y que no podrá ser modificado por el usuario. Así, la parte de la memoria principal cuyo contenido puede modificarse suele denominarse **RAM** (Random Acces Memory: memoria de acceso aleatorio; el tiempo de acceso no depende de la posición) y aquella que sólo puede leerse (y no modificarse) se denomina **ROM** (Read Only Memory).

Para solventar estas limitaciones de la memoria principal, en cuanto a su volatilidad (información perecedera) y su capacidad limitada, se emplean unos dispositivos periféricos, que constituyen la **Memoria Secundaria**, que permiten una capacidad muy superior (desde algunos Mb hasta los Gb) así como un almacenamiento permanente de la información (programas u otras informaciones), a no ser que se elimine o modifique de forma intencionada. Dado que el tiempo empleado en los procesos de lectura y escritura en la memoria secundaria es muy superior los que emplea la memoria principal, el ordenador siempre situará los programas en ejecución en la memoria principal. De esta forma, la memoria secundaria contribuye de manera fundamental a la versatilidad del ordenador, ya que podrá contener un elevado número de programas, con muy diversos propósitos, y podremos seleccionar en cada momento el que nos interesa llevar a la memoria principal para su ejecución.

4. Unidad Central de Proceso. Para poder realizar la ejecución de programas, en cada momento, el ordenador deberá realizar además las siguientes funciones:

a) Control.-. Debe establecerse cual es la dirección de memoria donde se encuentra el inicio del programa, y también cual es la dirección de la siguiente "instrucción elemental" a ejecutar. Debe establecerse la dirección en que se encuentra la información o datos que se van a procesar, o la dirección en la que se van a almacenar. Debe establecerse si la operación que se va ha realizar en una dirección de memoria es un proceso de lectura o escritura, si se está realizando una entrada o salida de información a través de la unidad de entrada/salida, y en cualquier tranferencia de información, debe establecerse cual es el elemento emisor, cual es el receptor, y en qué momento se debe realizar la transferencia.

En general, a partir de la lectura de cada una de las instrucciones de el programa que está ejecutándose en la memoria principal, deberá realizarse un control de todos los dispositivos que deben intervenir, estableciendo la función u operación que han de realizar y en que momento deben hacerlo.

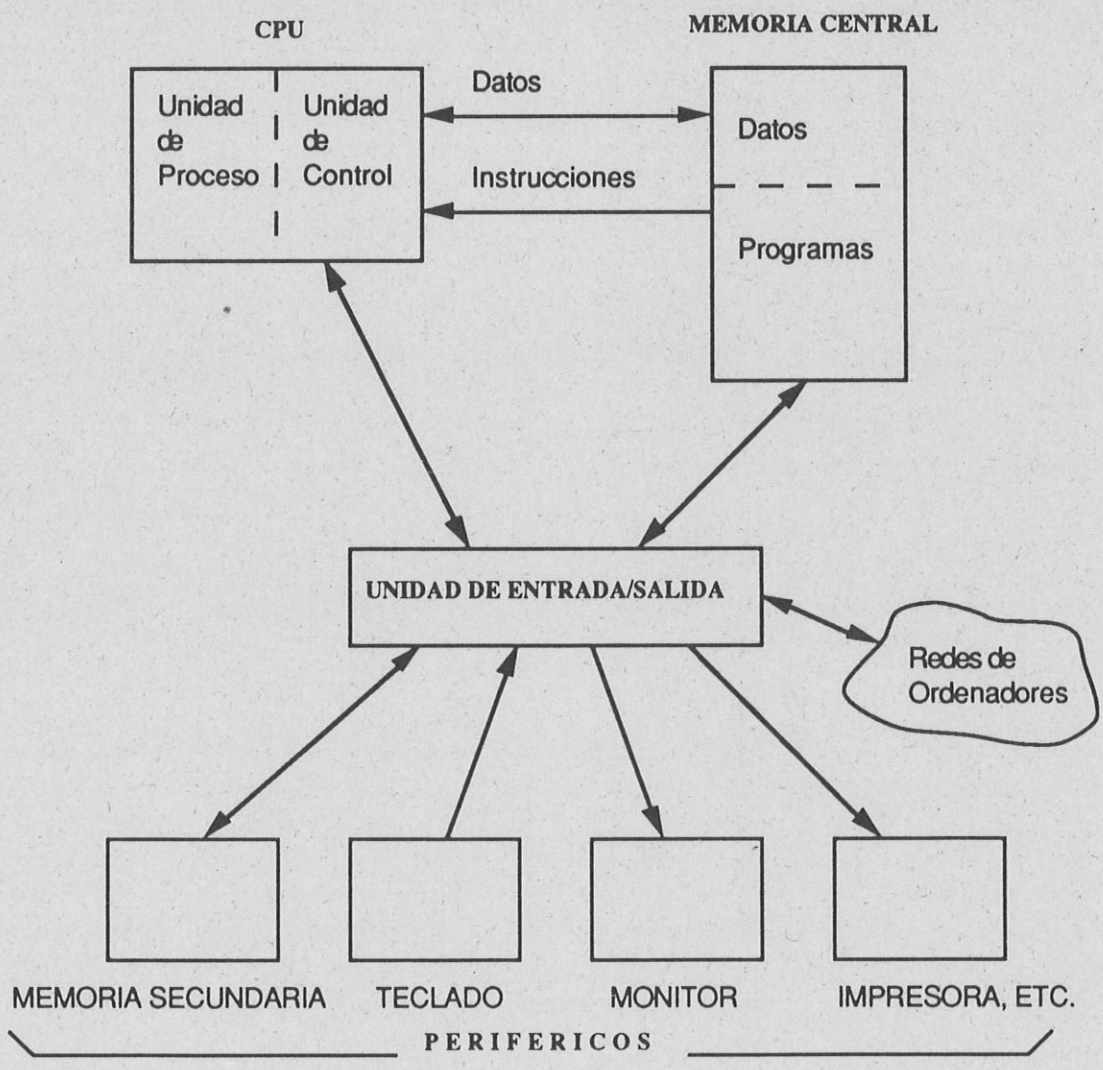
Todo ello para permitir una correcta coordinación de los distintos elementos del ordenador durante la ejecución de programas.

b) Proceso.-. Además, durante la ejecución de determinadas "instrucciones elementales", deben realizarse algunas operaciones básicas, con uno o más operandos, como son las operaciones aritméticas (realizadas en binario), o las operaciones de tipo lógico (determinar si se cumple o no determinada condición de igualdad o desigualdad, etc.).

Estas funciones de Control y Proceso primordiales las realizará la **Unidad Central de Proceso**, abreviada como **CPU** (de las iniciales en inglés). Formando parte de la CPU se encuentran la **Unidad de Control**, que realiza las funciones de Control (a), y la **Unidad de Proceso**, también denominada **Unidad Aritmético Lógica**, que reliza las funciones de Proceso (b). Con frecuencia, la CPU está fabricada en un único dispositivo físico denominado **Microprocesador**, que al igual que muchos otros dispositivos del ordenador, como la memoria central, utiliza la tecnología de Circuitos Integrados, basados en semiconductores (normalmente de silicio). Cada circuito integrado, también denominado **chip**, es de un tamaño muy reducido (de algunos milimetros cuadrados, contenido en una caja, con terminales exteriores o "pines", de pocos centímetros cadrados). Según sea la longitud de la "palabra" de información binaria que es capaz de procesar, tenemos microprocesadores de 8, 16 ó 32 bits (longitudes habituales). A

mayor longitud de palabra, mayor será el juego de "instrucciones elementales" que puede procesar, (que son propias de cada microprocesador), y mayor será la longitud de la información procesada en cada operación de lectura o escritura en la memoria principal, lo cual permite en general una mayor rapidez en la ejecución de programas, al implicar un menor número de accesos a memoria para tratar información de determinada longitud.

En el siguiente esquema se muestran los bloques principales del ordenador, ya comentados. Se indica la posible interconexión entre distintos ordenadores, que puede realizarse utilizando Redes de ordenadores.



1.4. ALGORITMO Y PROGRAMA

Vamos a analizar el proceso a seguir para obtener de la forma más adecuada un programa que nos permita realizar determinado tratamiento automático de información.

1. **Planteamiento**. En primer lugar, hemos de plantear en qué consiste el tratamiento de información que se desea realizar. Para ello, hemos de determinar cual es nuestra Información Primaria, las condiciones o hipótesis que se establecen sobre ella y los distintos casos que podrían darse. Debe determinarse cual es la Información Resultante que deseamos obtener, su significado y su relación con la información primaria. En un problema concreto, el planteamiento sería el enunciado del problema, con las hipótesis establecidas, y la descripción de los resultados que se desean.

2. **Análisis.** En este punto analizaremos el proceso a seguir a partir del planteamiento, con el objeto de conseguir un método para obtener la información resultante deseada, para los diversos casos que puedan darse. Habrán de determinarse las "estructuras de datos" (organización de la información a emplear en el proceso) que se van a necesitar, junto con la descripción y organización de las tareas a realizar (casos que puedan darse en el proceso, secuencia de operaciones a realizar, etc.). Será muy conveniente, en problemas de cierta complejidad, estructurar el análisis estudiando las tareas principales a realizar, para ir detallando éstas en etapas sucesivas hasta la descripcción del proceso en tareas "elementales".

El resultado del análisis es el **Algoritmo**: Descripción de las tareas a realizar para llegar a obtener la información resultante deseada en determinado tratamiento de información. Además, el número de tareas que supone la realización del algoritmo habrá de ser siempre un número finito, para cualquiera de los casos que éste contemple, ya que en caso contrario no podría llegar a obtenerse la información resultante. Para que un algoritmo sea de utilidad habrá de permitir su correcta interpretación por parte de cualquier persona, lo cual se consigue evitando toda posible ambigüedad en su descripción.

Para facilitar su interpretación y evitar toda posible ambigüedad, los algoritmos son descritos mediante representaciones normalizadas estandard. Una de las posibles representaciones, que

describiremos más adelante, es la representación gráfica mediante **Diagramas de Flujo** (también denominados Ordinogramas u Organigramas), que emplea una simbología estandard para la descripción de tareas.

3. Codificación del programa fuente. El diagrama de flujo del algoritmo diseñado durante el análisis no puede ser interpretado directamente por el ordenador. Para que éste pueda realizar el algoritmo diseñado, será necesario escribir un programa que reproduzca exactamente las tareas descritas en diagrama de flujo. Esto podrá hacerse utilizando un Lenguaje de Programación, el cual nos indica cómo podemos traducir las diferentes tareas del algoritmo en una serie de sentencias, que constituyen el programa correspondiente a dicho algoritmo y que habrá de introducirse en el ordenador. A este proceso se le denomina "Codificación del algoritmo en un Lenguaje de Programación". El texto del programa introducido en el ordenador será el Programa Fuente.

1.5 FICHEROS. SISTEMA OPERATIVO.

1. Ficheros.

La información que constituye el texto de un programa fuente tiene un significado por sí misma, ya que tiene un propósito determinado. Para poder utilizar distintos programas en la memoria secundaria (normalmente un disco magnético), debe asignarse a cada programa un nombre que sirva para su identificación. Además, como pueden existir programas fuente escritos en distintos lenguajes de programación, se utilizará una identificación adicional, que llamaremos extensión, y que será distinta para cada lenguaje. Esta extensión suele contener hasta 3 letras. A cada bloque de información designado de esta forma y almacenado en la Memoria Secundaria se le denomina **Fichero.** Los programas fuente son sólo un caso particular de fichero, cuya información puede ser visualizada en la pantalla. Otros casos de ficheros son, por ejemplo, los programas ejecutables (con el mismo nombre que el programa fuente y extensión EXE, y que habrán de ser trasladados a la memoria central para su ejecución), ficheros de datos, documentos (texto), ficheros de comandos, etc. Cada fichero queda designado así por su **Nombre** y **Extensión**, separadas por un punto. Por ejemplo:

CALCUL.PAS	Programa fuente en lenguaje Pascal.
CALCUL.OBJ	Programa objeto (resultado de "compilación")
CALCUL.EXE	Programa ejecutable (resultado del "montaje").
CALCUL.C	Programa fuente en lenguaje C.
PERSONAL.DAT	Fichero de datos de personal.
INFORME.TXT	Fichero de texto.

2. Sistema Operativo.

Para poder realizar operaciones básicas con los ficheros (listado o clasificación de ficheros existentes, creación, visualización, modificación, borrado, ejecución de programas ejecutables, etc.), y para poder utilizar todos los recursos que el ordenador pone a nuestra disposición (desde su puesta en marcha), existe un conjunto de programas que deben estar disponibles en el ordenador para su uso, y que constituyen el **Sistema Operativo**. El uso de estos programas se realiza mediante una serie de Comandos (Palabras clave, normalmente en inglés, con sus propias reglas sintácticas), que son propios de cada sistema operativo, y que permiten realizar las funciones u operaciones necesarias. Por ejemplo el comando

HELP del VMS nos facilita la explicación (en inglés) de cómo se utilizan todos sus Comandos y posibilidades de uso. El comando DIR permite ver la lista de ficheros existentes en el disco en algunos sistemas operativos, como son el DOS (Disk Operating Sistem) o el VMS.

3. Editores y Compiladores.

Cada sistema operativo posee al menos un **Editor,** que permite realizar con relativa facilidad la creación o modificación de ficheros de texto, como es el caso de los programas fuente, documentos, etc.

En algunos sistemas operativos, como el VMS (Virtual Memory System), pueden coexistir distintas "versiones", resultado de sucesivas modificaciones de un mismo fichero con igual nombre y extensión. Estas se distinguen entre sí por el número de versión:

CALCUL.PAS;1 CALCUL.PAS;2 etc.

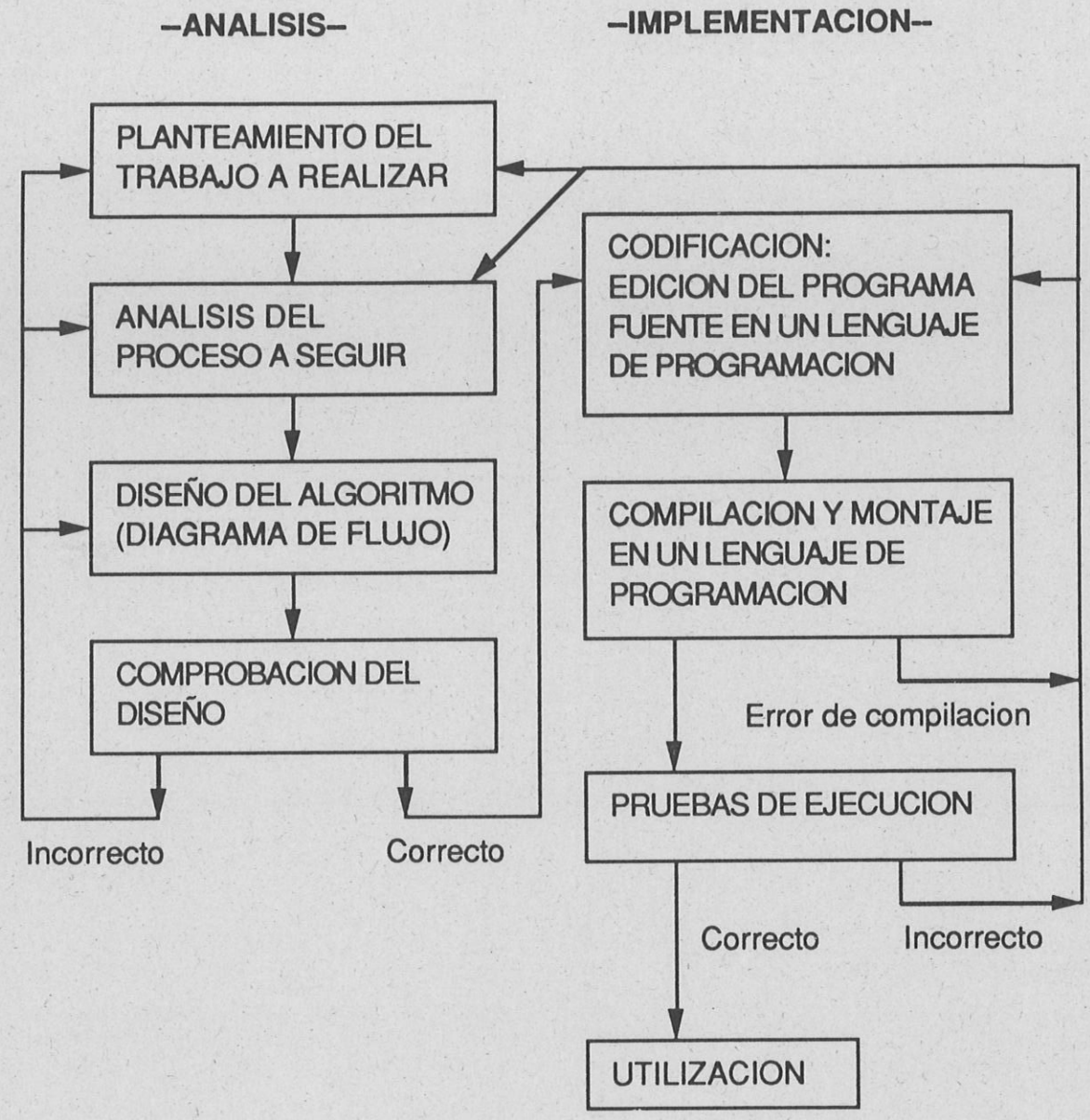
En el sistema operativo MS-DOS, suele guardarse la última verisón de un fichero modificado, a la cual le asigna la extensión .BAK:

CALCUL.BAK CALCUL.PAS

Para poder obtener programas ejecutables a partir de los programas fuente se debe utilizar un **"Compilador"**, que es específico de cada lenguaje de programación. Al proceso de traducción para obtener el programa ejecutable se le denomina **Compilación.** Algunos compiladores poseen herramientas (programas de utilidad) que facilitan el proceso de programación: Editor propio, ejecución paso a paso, tratamiento y depuración de errores durante la "traducción" del programa fuente.

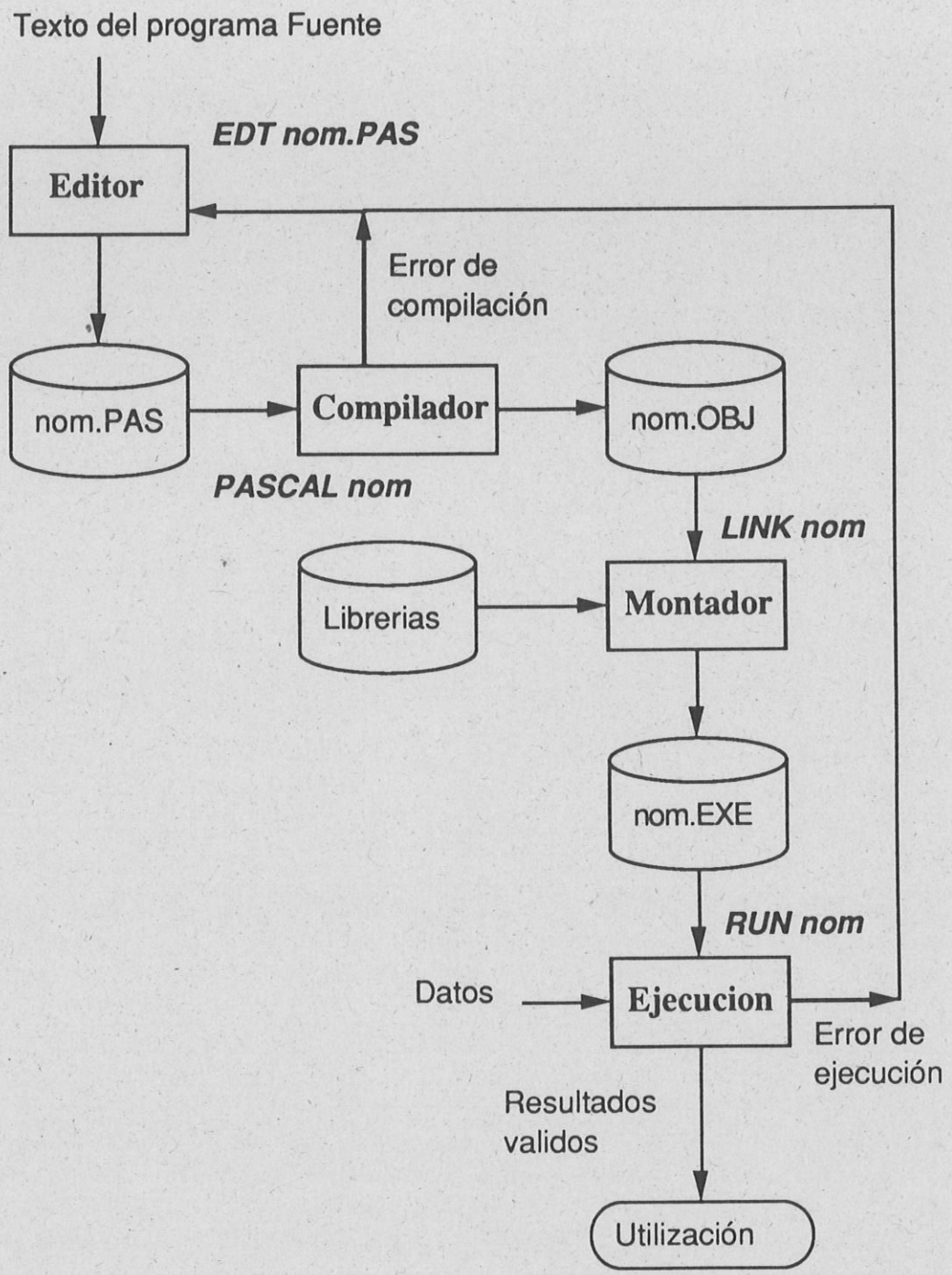
1.6. PROCESO DE EDICION, COMPILACION Y EJECUCION DE PROGRAMAS

A continuación se indican cuales son las fases que en general se seguirán durante el proceso de obtención de un programa ejecutable. En una primera etapa, Análisis, se realiza el diseño del algoritmo. Una vez se han realizado las comprobaciones adecuadas del mismo, se pasa a la siguiente fase, Implementación, donde se edita el programa fuente, para después compilarlo y ejecutarlo, sometiéndolo a diversas pruebas de ejecución para comprobar los distintos casos que se hayan previsto en el programa. En todas estas etapas, puede ser necesario volver a alguna etapa anterior, para revisar y corregir posibles errores producidos, por ejemplo, durante la compilación o la ejecución del programa.



Veamos ahora, con más detalle, el proceso a seguir en el caso de utilizar un compilador de VAX-PASCAL. Como puede verse, el fichero resultante de la compilación se denomina nom.OBJ (programa objeto), el cual aún no es el ejecutable. Para obtener el programa ejecutable nom.EXE debe utilizarse el Montador o "Linker", el cual anexa al programa objeto determinados bloques de programa (librerías) de que dispone el compilador y que son utilizados por nuestro programa cada vez que se requiere el uso de funciones o subprogramas propias del lenguaje Pascal. De esta forma se obtiene el programa ejecutable nom.EXE, el cual será llevado a memoria para su ejecución mediante el comando RUN nom.

PROCESO DE EDICION, COMPILACION MONTAJE Y EJECUCION EN VAX-PASCAL:



En el proceso anterior, se han indicado dentro de un cilindro (que representa a la información en el disco) los ficheros que se generan en cada etapa del proceso. Se han indicado en cursiva los comandos utilizados del Sistema Operativo VMS, en modo Interactivo.

Compilación mediante colas de Batch:

Las etapas de compilación y montaje suelen realizarse empleando procesos por "lotes" (batch) que son realizados por la CPU según el orden y la prioridad que ocupan en una "cola" de espera. El uso de procesos batch permite que, para realizar cualquier otro proceso, (edición, ejecución de otro programa o comando, etc.) no debamos esperar a que termine la compilacion del programa. El sistema operativo no avisa con un mesaje cuando ha terminado el proceso batch. Para realizar fácilmente el uso de procesos batch debemos escribir un fichero que contiene comandos del sistema operativo (extensión .COM), el cual puede ejecutarse para enviar el proceso a la cola seleccionada. Veamos un ejemplo de cómo puede hacerse la compilación y montaje de programas Pascal con facilidad:

Editamos un fichero con nombre ENVIA.COM :
EDT ENVIA.COM, con el siguiente texto:

```
$ INQUIRE NOM "NOMBRE DEL PROGRAMA ? "
$ INQUIRE COLA "NOMBRE DE LA COLA ? "
$ OPEN /WRITE FICHERO BATCH.COM
$ WRITE FICHERO "$ SET DEF [.directorio]
$ WRITE FICHERO "$ PASCAL "'NOM
$ WRITE FICHERO "$ LINK "'NOM
$ CLOSE FICHERO
$ SUBMIT/QUE='COLA BATCH.COM
```

(Salir del editor con: CTRL-Z y EXIT)

Donde [.directorio] es el nombre del directorio donde tenemos el programa fuente, con extensión .PAS: Nombre.PAS
Si el programa fuente está en el directorio principal (Directorio Raiz, en el que entramos siempre por defecto) no debe escribirse la cuarta línea del programa anterior.

De esta forma enviaremos a la "cola", mediante el comando SUBMIT/QUE, un fichero de comandos denominado BATCH.COM que contiene los procesos de compilación y montaje del programa deseado.

Una vez creado el fichero del programa fuente: Nombre.PAS, ejecutamos el programa de comandos anterior mediante: @ENVIA
Hemos de entrar el Nombre del programa (Nombre, sin extensión, pues asumirá que es .PAS), y nombre de la Cola que utilizamos, por ejemplo LENTA). El programa genera el fichero BATCH.COM y lo envia a la cola seleccionada. Podemos realizar otras tareas con el

ordenador mientras éste atiende a los procesos pendientes existentes en la cola. Cuando aparece el mensaje:

Job BATCH (queue LENTA, entry xxx) completed

entonces se habrá generado el fichero ejecutable Nombre.EXE, y podremos ejecutar el programa mediante: RUN Nombre
En este caso, la ejecución ya no se realizará en modo de proceso batch, sino en modo Interactivo, que es el habitual para la ejecución de programas. Podrá realizarse la ejecución en Batch si la entrada y salida de datos se realiza mediante ficheros de entrada/salida, lo cual puede ser útil para programas de cálculo complejos que requieren mucho tiempo de ejecución.

En el caso de utilizar Turbo-Pascal, entraremos en el "entorno" de TurboPascal (programa basado en menús que gestiona el uso de este compilador, con un editor propio) mediante TURBO. En la barra de menú seleccionamos: File - New. Una vez editado el programa fuente, aún denominado NOMANE.PAS, podemos darle nombre mediante: ALT-F-Write to, y entramos nuevo nombre (por ejemplo PRUEBA, asume extensión .PAS por defecto).
A continuación lo podemos ejecutar mediante CRTL-F9.
Si quemos obtener un fichero ejecutable PRUEBA.EXE, debemos seleccionar en la barra de menú de Compilación el destino a disco: ALT-C-Destination, y cambiar a Disk (si estaba en Memory) pulsando Return, y a continuación compilar mediante: ALT-F9.
En ese momento se crea en el directorio por defecto el fichero PRUEBA.EXE. Al salir del entrono de TurboPascal, mediante ALT-X, podremos ejecutar el programa simplemente mediante: PRUEBA.

2. ALGORITMOS

En este capítulo se describirán conceptos básicos de algoritmos y su codificación en leguaje Pascal.

2.1. REPRESENTACION. DIAGRAMAS DE FLUJO.

La representación de algoritmos mediante diagramas de flujo tiene por objeto sistematizar en lo prosible el proceso de codificación de programas a partir de la descripción de un algoritmo. Recordemos brevemente los conceptos de algoritmo y programa:

Algoritmo: Descripción no ambigua de las tareas a realizar en determinado tratamiento de información (con un número finito de tareas) para obtener la información resultante deseada.

Programa: Es la traducción (codificación) del algoritmo en un lenguaje, definido mediante una reglas prefijadas, resultando de ello un conjunto ordenado de sentencias, definidas en dicho lenguaje, que permiten al ordenador realizar el proceso descrito en el algoritmo.

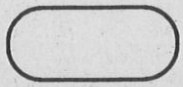
Un diagrama de flujo (también denominado Organigrama u Ordinograma) es una representación gráfica de un algoritmo. La simbología internacional utilizada para estos diagramas es la definida por ANSI (American National Standard Institute). En la página siguiente se indican los símbolos más importantes.

La información requerida para describir la tarea a realizar se indica en el interior de cada símbolo. En el caso del condicional, se indican fuera del símbolo los caminos posibles, con V (verdadero) o F (falso) en el vértice correspondiente.

Las líneas terminadas en flecha interconectan a los símbolos entre sí, indicando su orden de realización. Debe evitarse en el diagrama cualquier cruce de líneas, y en los casos en que no pueda evitarse se usarán los conectores, en el menor número posible.

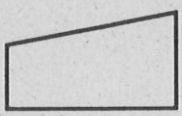
En la descripción de procesos complejos, es muy conveniente el uso de bloques de llamada a subprograma, lo cual permite una mayor claridad al analizar el proceso descrito en los diagramas.

SIMBOLOGIA ANSI DE DIAGRAMAS DE FLUJO:

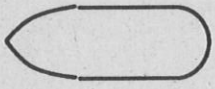


Inicio y fin del diagrama

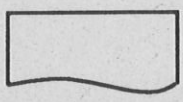
Indicación del orden de realización de tareas, conectando entre sí los símbolos del diagrama



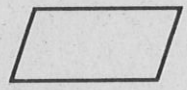
Entrada de información por teclado



Salida de información por pantalla

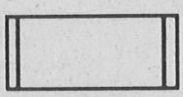


Salida de información por impresora

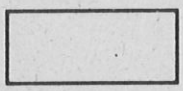


Entrada/salida de información mediante ficheros

Condicional: Según sea verdadera o falsa la condición indicada, el proceso continúa por el camino correspondiente



Llamada a un "subprograma". Sustituye a otro diagrama, descrito aparte.



Cualquier tarea, en general, no descrita por otro símbolo.

Conector entre puntos del diagrama de una misma página

Conector entre puntos del diagrama de páginas diferentes

2.2. VARIABLES. TIPOS DE DATOS.

Al utilizar el ordenador en la ejecución de cualquier programa, todos los datos (información primaria), resultados intermedios y resultados finales (normalmente), deberán estar almacenados en memoria, en forma de ciertas unidades de información, que sean de utilidad en la descripción del algoritmo a realizar.
Para acceder a cada una de estas unidades de información habrá que hacer referencia, de alguna forma, a la dirección de memoria donde se encuentra. Para facilitar esta labor, a cada dirección de memoria que contenga una unidad de información a utilizar se le asocia un nombre (que normalmente guardará cierta relación con el significado de dicha unidad de información), de forma que al referenciar dicho nombre (en lugar de una dirección en binario) obtenemos la información que contiene la memoria en la posición correspondiente.

Se denomina **Variable** a cada una de las unidades de información utilizadas en un algoritmo, o programa, cuyo valor puede cambiar en el transcurso del proceso. Una variable quedará determinada por:

Nombre: Que se empleará para localizar la información deseada en la memoria, y distinguirla de otras informaciones de la memoria.
Tipo: Que indica el conjunto de posibles valores que la variable puede contener (números enteros, reales, etc.).

Durante la ejecución del programa cada variable puede contener un **Valor:** Información que podrá cambiar en el transcurso del proceso. Inicialmente el valor de cualquier variable es indefinido.

El tipo de dato asociado a cada variable determina el número de bytes que ocupa en memoria la información que contiene cada variable, y los valores que puede contener. Así, los tipos de datos estandares más simples del lenguaje Pascal son:

Integer:	números enteros
Real:	números reales
Char:	caracteres (un sólo caracter en cada variable: pueden ser letras, dígitos decimales, símblos)
Boolean:	valores booleanos (TRUE=cierto, FALSE=falso)

Estos tipos de demoninan *escalares* (cada variable coniene sólo un único valor), en contraposición a los tipos *estructurados*, que permiten el acceso a un conjunto de valores.

Sobre una variable cualquiera pueden realizarse dos operaciones básicas:

Lectura: Permite obtener el valor que dicha variable contiene. Para ello basta con hacer referencia al nombre de la variable en el momento que precisemos obtener su valor. En esta operación, el valor de la variable no se modifica. El caso habitual de lectura se presenta al emplear variables en cualquier expresión, para poder evaluar el valor resultante. En los casos en que el tipo de dato lo permita, el valor resultante de la expresión, o el valor de la variable que se desee, podrá visualizarse en pantalla mediante una operación de salida por pantalla. También podrá almacenarse en un fichero mediante una operación de salida mediante ficheros:

Escritura: Permite definir un nuevo valor en la variable, o modificar el valor que ya tenía definido. Esto se podrá realizar (aunque no siempre) de dos formas:

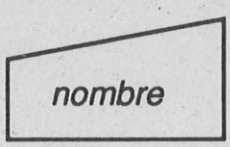
a) Mediante una operación de **Asignación,** la cual se representa de la siguiente forma en el diagrama de flujo:

La correspondiente codificación en Pascal es: *nombre* := *expresion*
Donde *nombre* es el nombre de la variable sobre la cual se realiza la asignación, y *expresion* puede ser un valor constante, otra variable, o cualquier otra secuencia de "operaciones" (descritas mediante operadores) con constantes o variables que nos permita obtener un valor resultante, de acuerdo con la sintaxis del lenguaje. En cualquier caso, el valor resultante habrá de ser del mismo tipo que la variable sobre la que se realiza la asignación.
El operador de asignación es el que se indica a la derecha de *nombre.*

La asignación se realiza mediante el proceso siguiente:
1.- Se evalúa la expresión: Cada variable que aparece en la expresión, implica una operación de lectura del valor de la misma, que será el que interviene en la realización de las operaciones descritas en la expresión.

2.- El valor resultante de la expresión pasa a ser el nuevo valor de la variable situada a la izquierda del operador de asignación, con lo que se hace una operación de escritura sobre dicha variable. El valor que contenía anteriormente se pierde.

b) En los casos en que el tipo de dato lo permita, mediante una operación de **Entrada de Información** (mediante el teclado) sobre dicha variable, que tiene la siguiente representación en el diagrama de flujo:



La correspondiente codificación en Pascal es: READ(*nombre*)

Donde *nombre* es el nombre de la variable sobre la cual se realiza la operación de entrada.
En este caso, al llegar a este punto, el proceso queda a la espera de que se introduzca un valor mediante el teclado, del mismo tipo que la variable, y de acuerdo con la sintaxis del lenguaje. En el momento en que se realice dicha entrada, la variable toma el valor introducido y el proceso continúa. El valor asignado no podrá ser una expresión, habrá de ser un valor constante concreto. El valor que contenía la variable anteriormente se pierde.
Esta operación de entrada es la que permite que un mismo proceso (algoritmo, programa) pueda realizarse con diversidad de informaciones primarias, debiendo por tanto contemplarse dicha diversidad en el diseño del algoritmo.
También podría realizarse la entrada de información mediante ficheros, que se codificaría en lenguaje Pascal mediante la sentencia: READ(*nombre_del_fichero,nombre*)

Puede realizarse la entrada de información sobre varias variables en la misma operación de entrada, indicando sus nombres, dentro del mismo símbolo de entrada, separados por comas. En este caso, debe entrarse por teclado los valores de las correspondientes variables en el mismo orden (en caso de variables Integer o Real estos valores estarán separados por al menos un espacio en blanco al escribirlos mediante el teclado).

Expresiones.

Las expresiones que pueden utilizarse (por ejemplo, en una operación de asignación) constan de una secuencia de operandos y operadores, sobre los que se establecen unas reglas de prioridad. Los operadores (por ejemplo: +, -, *, /) deben utilizarse siguiendo ciertas reglas en cuanto al tipo de datos de los operandos. La evaluación de la expresión da como resultado un valor de cierto tipo de datos, por ejemplo Integer, Real, o Boolean. Los operandos pueden ser constantes, variables o expresiones, del tipo adecuado al operador utilizado. Se utilizan los paréntesis para cambiar las prioridades en la realización de las operaciones. Una expresión también puede incluir únicamente una variable, o un valor constante. Por ejemplo, son expresiones:

E1 = Expresión con resultado Real:.....(A+B)*(4-A/(X+1))*2
E2 = Expresión formada por sólo una variable:..... B
E3 = Expresión formada por sólo una constante:.. 2.5

Los operadores aritméticos tienen el siguiente orden de prioridad:

Mayor prioridad: * /
Menor prioridad: + -

A igual prioridad, se realizan de siguiendo el orden que ocupan en la expresión, de izquierda a derecha. En cualquier caso, si se indican paréntesis, se realizan en primer lugar los paréntesis más internos de la expresión, para continuar con los siguientes.

De esta forma, como ejemplo, la expresión e1 se realizará en el siguiente orden (se indican con ki los resultados intermedios):

k1:= X+1
k2:= A+B
k3:= A/k1
k4:= 4-k3
k5:= k2*k4
Resultado= k5*2

El valor resultante sustituye a la expresión, allí donde se utilice, siendo inaccesibles los resultados intermedios (k1,.. k5).

Se emplearán expresiones con resultado de tipo Boolean cuando utilicemos un símbolo de Condicional, visto en el apartado anterior. Si utilizamos operadores relacionales y expresamos con e1, e2, ... cualquier conjunto de expresiones, (cuyo valor resultante pertenezca a un tipo de dato que tenga definida una relación de orden o que permita el uso de estos operadores), podremos construir expresiones booleanas sencillas, que denominaremos eb, mediante el uso de estos operadores relacionales:

eb1:	e1 < e2	Resultado TRUE si el valor de e1 es menor que e2 y FALSE en caso contrario. Análoga interpretación para el resto de operadores:
eb2:	e1 <= e2	TRUE si e1 es menor o igual que e2.
eb3:	e1 = e2	TRUE si los valores de e1 y e2 coinciden.
eb4:	e1 <> e2	TRUE si los valores de e1 y e2 son distintos.
eb5:	e1 >= e2	TRUE si e1 es mayor o igual que e2
eb6:	e1 > e2	TRUE si e1 es mayor que e2

Además podemos construir expresiones booleanas más complejas utilizando expresiones booleanas sencillas, como las descritas antes, y empleando operadores lógicos:

eb1 AND eb2	Resultado TRUE cuando ambas expresiones eb1 y eb2 sean TRUE simultáneamente. Resultado FALSE en caso contrario.
eb1 OR eb2	Resultado FALSE cuando ambas expresiones eb1 y eb2 sean FALSE simultáneamente. Resultado TRUE en caso contrario.
NOT eb	Resultado TRUE cuando eb sea FALSE. Resultado FALSE cuando eb sea TRUE.

Los operadores booleanos vistos tienen el siguiente orden de prioridad (orden en que se ralizan las operaciones):

Mayor prioridad:	NOT
	AND, OR
Menor prioridad:	Operadores relacionales

Debido a ello, si queremos realizar en primer lugar los operadores relacionales, antes que los operadores lógicos, las expresiones booleanas sencillas (que solo utilizan operadores realcionales) se indicarán entre paréntesis. Por ejemplo:

Expresión correcta:

(X > 0) AND (Y > 0) — Resultado TRUE cuando ambas variables X, Y sean simultáneamente mayores que 0.

Expresión incorrecta:

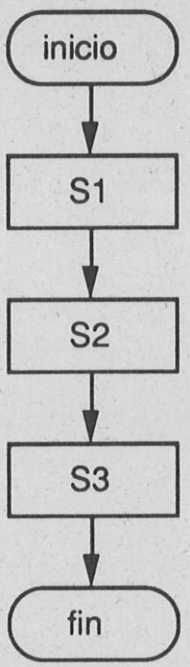
X > 0 AND Y > 0 — Ya que se intentaría realizar en primer lugar la operación 0 AND Y, que no tiene sentido por no ser 0 ni Y expresiones booleanas.

2.3. ESTRUCTURAS ALGORITMICAS BASICAS.

Utilizando los símbolos estandares para la representación de algoritmos, definiremos unas estructuras algorítmicas básicas, de forma que empleando únicamente dichas estructuras pueda representarse cualquier algoritmo.
Cada una de estas estructuras posee un único inicio y un único final, e indicaremos para cada una la codificación Pascal correspondiente.
Una tarea general cualquiera del diagrama la denominaremos **Sentencia**, que representaremos mediante **S,** la cual posee un único inicio y final. También serán sentencias las propias estructuras básicas que se describen, ya que también poseen un único inicio y final.

1. Estructura secuencial.

Esta estructura se define como **Sentencia Compuesta**, del lenguaje Pascal.
La estructura incluye un conjunto de sentencias, que son realizadas secuencialmente en el orden que se indica. Cada sentencia se inicia cuando ha finalizado la sentencia anterior. Todas las sentencias de la estructura se sealizan una sola vez. Denominaremos **Si** a las sentencias de la estructura (i= 1,2,..n). Para n=3 la representación de esta estructura y su codificación en Pascal son:



Codificación en Pascal:

a)

```
BEGIN S1; S2; S3 END
```

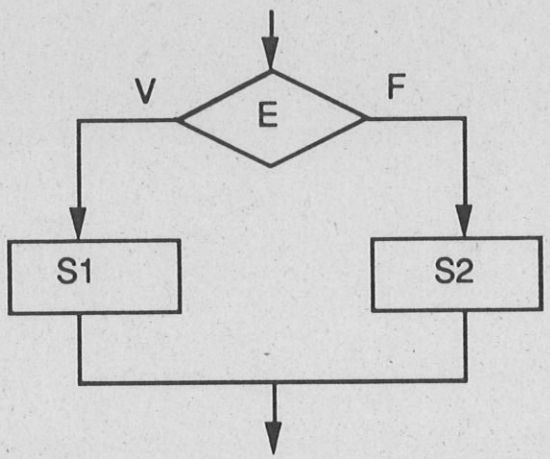
b)

```
BEGIN
    S1;
    S2;
    S3
END
```

En la codificación en Pascal se indica el inicio y final de la secuencia de sentencias, o sentencia compuesta, con las palabras BEGIN y END respectivamente, y las diferentes sentencias se separan entre sí mediante el signo [;] que actúa de delimitador (separador) y que no se emplea antes de la primera ni después de la última, ya que en dicho caso no separa entre sí a dos sentencias de la secuencia. Las dos descripciones (a) y (b) de codificación Pascal son equivalentes. En la segunda, después de cada sentencia se han intercalado saltos a la línea siguiente y un cierto margen para obtener una codificación más fácilmente legible (ésta segunda forma será la utilizada habitualmente).

2. Estructura alternativa.

Esta estructura se define como **Sentencia IF** del Pascal.
La estructura incluye dos sentencias, **S1** y **S2**, y un condicional, que está definido por una expresión booleana **E**. Según sea TRUE (Verdadero) o FALSE (Falso) el resultado de dicha expresión booleana, se realiza una u otra sentencia. La representación de esta estructura y su codificación en Pascal son:



Codificación en Pascal:

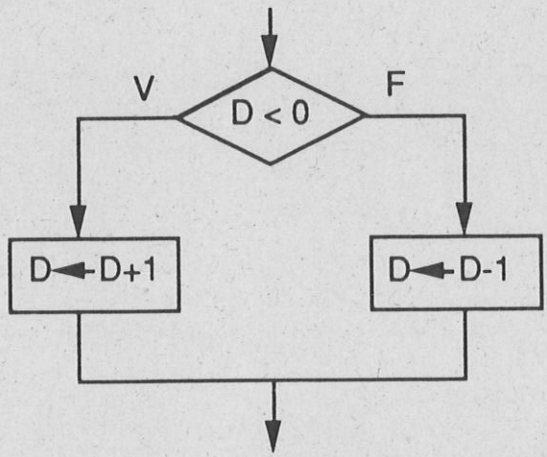
```
IF E THEN S1 ELSE S2
```

o bien:

```
IF E
   THEN S1
   ELSE S2
```

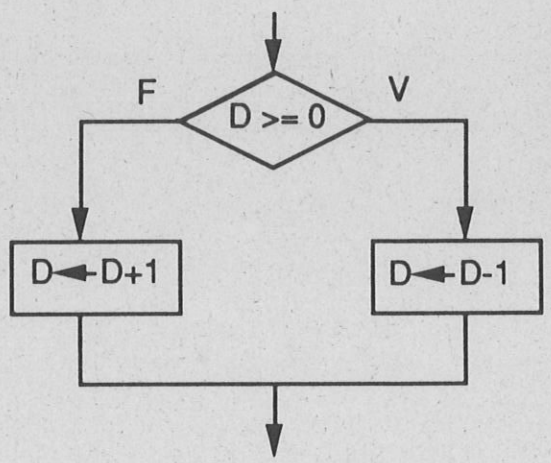
El proceso seguido es:
1. Se evalúa la expresión booleana E.
2. Si el resultado de E es TRUE, se realiza la sentencia S1 y finaliza la estructura.
3. En caso contrario (el resultado de E es FALSE) se realiza la sentencia S2 y finaliza la estructura.

Ejemplo:



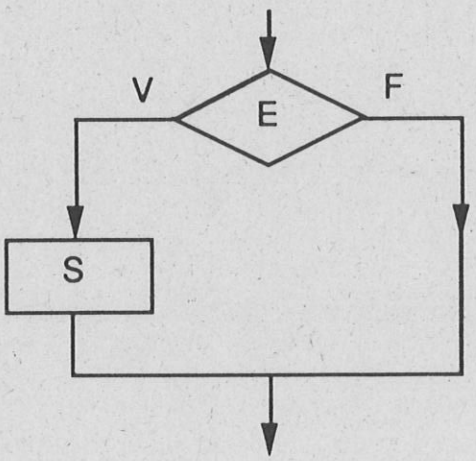

```
IF D < 0
   THEN D := D+1
   ELSE D := D-1
```

Una estructura o sentencia IF equivalente se obtendrá cambiando la expresión booleana E por la contraria de E (en general NOT E), e intercambiando la situación de las alternativas Verdadero (TRUE) y Falso (FALSE) en el símbolo condicional. Para el ejemplo anterior, la estructura equivalente, y su codificación Pascal, serían:



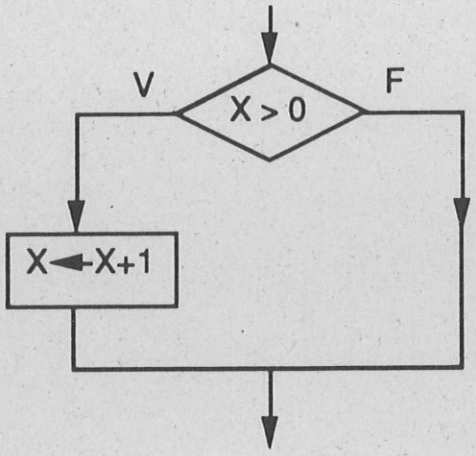

```
IF D >= 0
   THEN D := D-1
   ELSE D := D+1
```

Si en el caso de que la expresión booleana sea FALSE no existe ninguna sentencia a realizar (S2 = sentencia nula, no realiza ninguna acción), la sentencia IF admite la simplificación de omitir la parte ELSE en su codificación en Pascal:



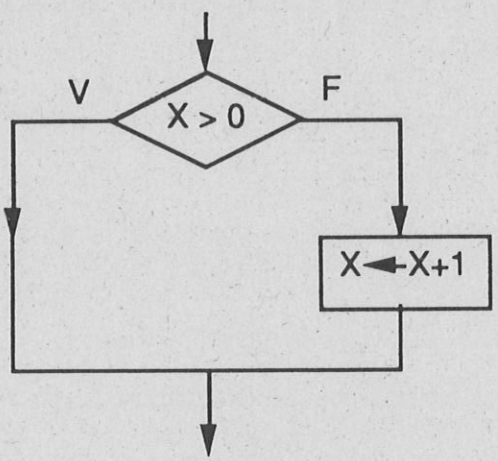

```
IF E THEN S
```

Ejemplo:




```
IF X > 0
   THEN X := X+1
```

Si fuera S1 la sentencia nula, y E la expresión booleana, entonces puede obtenerse la estructura equivalente, para la cual S2 se realizará cuando NOT E sea cierto, empleando también la simplificación de la sentencia IF. Un ejemplo de ello es el siguiente:




```
IF X <= 0
   THEN X := X+1
```

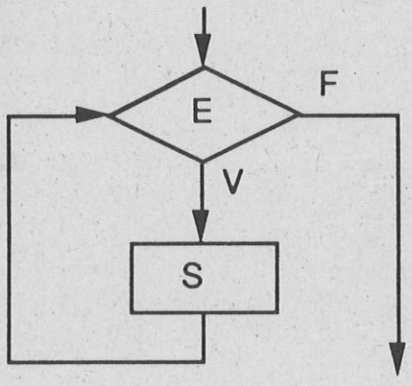
3. Estructura Repetitiva.

A diferencia de las estructuras anteriores, en las que cada sentencia podía realizarse sólo una vez (o ninguna), en este caso la sentencia (o sentencias) incluidas en estructuras repetitivas podrán realizarse, en general, más de una vez, ya que el proceso vuelve a algún punto anterior a la sentencia, dependiendo del resultado de cierta expresión booleana.

Existen dos tipos de estructura repetitiva, cada una de las cuales define una sentencia del Pascal. Son las siguientes.

1) Sentencia **WHILE** del Pascal: Mientras cierta condición sea cierta, realizar la sentencia incluida en la estructura.
La estructura incluye un condicional, definido por una expresión booleana **E,** y una sentencia. **S**.

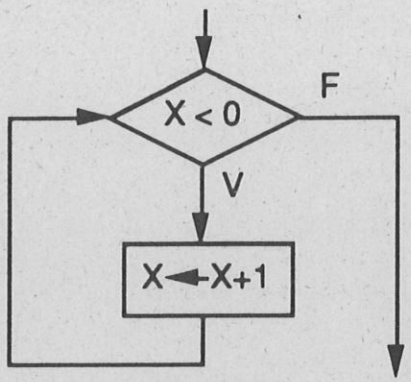
La representación de esta estructura y su codificación en Pascal son:



WHILE E DO S

El proceso seguido es el siguiente:
1. Se evalua la expresión booleana E.
2. Si el resultado de E es FALSE, finaliza la estructura.
   Si el resultado de E es TRUE, se realiza la sentencia S y el proceso
   continúa de nuevo en el punto 1.

Ejemplo:



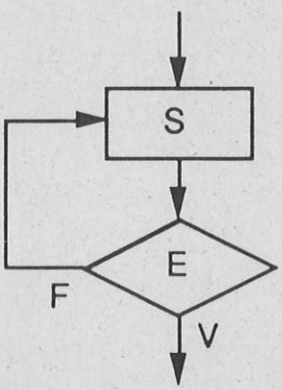
WHILE X<0 DO X:=X+1

En el caso en que S corresponda a una sentencia compuesta, por ejemplo de 3 sentencias S1, S2 y S3, se indicará la sentencia compuesta mediante la estructura secuencial ya conocida. La codificación en Pascal sería, en este caso, la siguiente:

```
WHILE E DO
BEGIN
   S1;
   S2;
   S3
END
```

2) Sentencia **REPEAT** del Pascal: Repetir la sentencia incluida en la estructura hasta que cierta condición sea cierta.
Al igual que la estructura anterior, ésta incluye un condicional, definido por una expresión booleana **E,** y una sentencia. **S**.

La representación de esta estructura y su codificación en Pascal son:

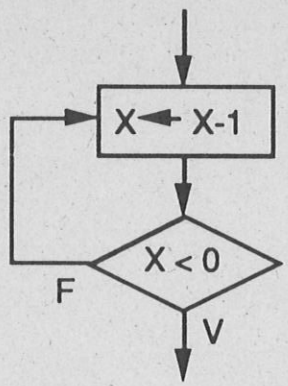


REPEAT S UNTIL E

El proceso seguido es el siguiente:

1. Se realiza la sentencia S.
2. Se evalúa la expresión booleana E.
3. Si el resultado de E es TRUE, finaliza la estructura.
   Si el resultado de E es FALSE, el proceso continúa de nuevo en el punto 1.

Ejemplo:




```
REPEAT
  X := X-1
UNTIL X<0
```

En el caso en que S corresponda a una sentencia compuesta, por ejemplo de 3 sentencias S1, S2 y S3, se indicará la sentencia compuesta mediante la estructura secuencial. Sin embargo, a diferencia de la sentencia WHILE, y como única excepción, en este caso no es preciso el uso de las palabras BEGIN y END para delimitar el inicio y final de la secuencia de sentencias. Esta misión la suplen las palabras REPEAT y UNTIL, que también delimitan la secuencia de sentencias. Esto no ocurría en la sentencia WHILE (sólo aparece en ella la palabra DO al inicio de la sencuencia). La codificación en Pascal sería, en este caso, la siguiente:

```
REPEAT
  S1;
  S2;
  S3
UNTIL  E
```

2.4. TRAZA DE ALGORITMOS. EJEMPLOS.

1. Ejemplos de algoritmos elementales. Codificación en Pascal.

Con las estructuras básicas analizadas en el apartado anterior, podemos analizar el funcionamiento de algoritmos sencillos y obtener una primera codificación en Pascal de los mismos.

Ejemplo 1:
Máximo comun divisor de dos números:

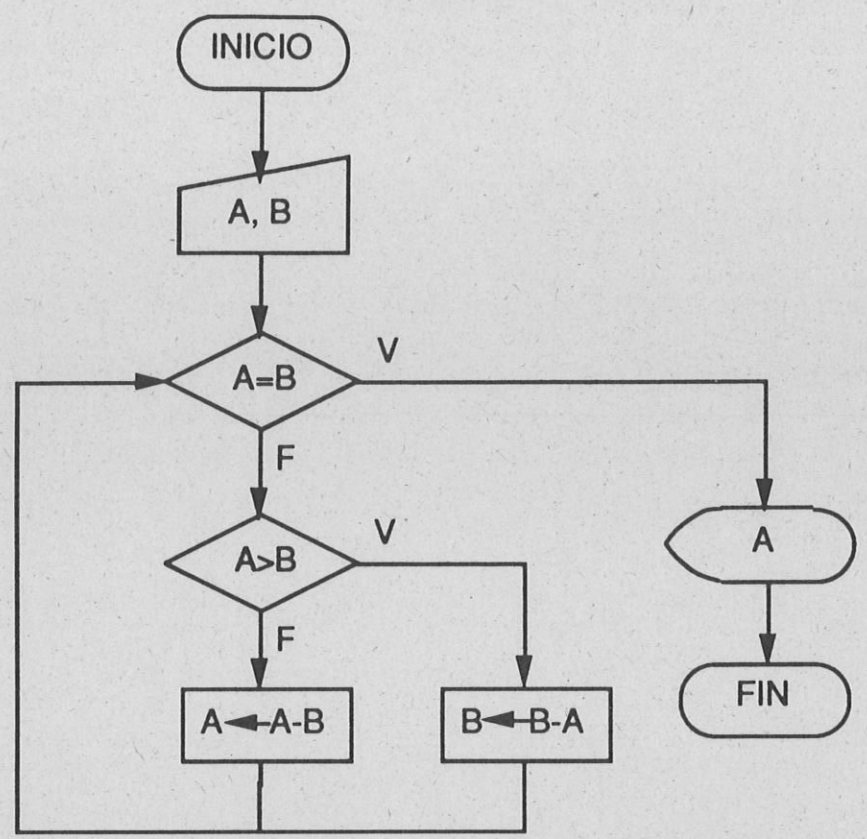
Dados dos valores A y B, el maximo común divisor de ambos puede obtenerse mediante el siguiente proceso (algoritmo de Euclides):

1. Tomar dos números A y B enteros positivos.
2. Si A y B son iguales, escribir el valor de A y finalizar.
3. Si son distintos:
   Cambiar el valor del mayor de los dos números A y B por el resultado de la diferencia entre el mayor y el menor, y volver al punto 2.

Las operaciones necesarias para realizar el punto 3 pueden a su vez describirse con más detalle de la forma siguiente:

3.1. Si A es mayor que B: Cambiar el valor de A por el valor de A-B y volver al punto 2.
3.2. En caso contrario (si es A menor que B): Cambiar el valor de B por el valor de B-A y volver al punto 2.

La descripción del proceso anterior puede representase mediante un organigrama en la forma siguiente:



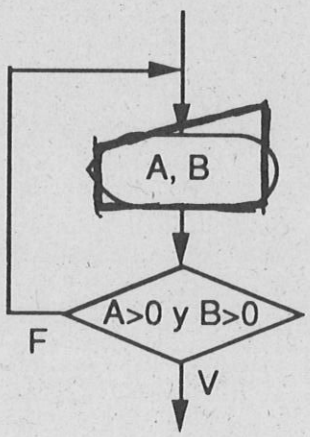
A partir del algoritmo anterior, puede obtenerse un programa codificado en lenguaje Pascal. Todo programa consta de una cabecera, en la que se le da un nombre al programa, y (normalmente) una parte de declaraciones, en la cual se indican, precedidos de la palabra VAR, los nombres de las variables empleadas y el tipo de dato correspondientes. Las sentencias del programa se inician siempre con la palabra BEGIN y finalizan con END, seguido de un punto. El programa Pascal podrá ser el siguiente:

```
PROGRAM MCD ( INPUT, OUTPUT );
VAR A, B : INTEGER;
BEGIN
 WRITE('ENTRAR DOS NUMEROS ENTEROS ');
 READ(A,B);
 WHILE A<>B DO
     IF A>B THEN A:=A-B
            ELSE B:=B-A;
 WRITE('EL MAXIMO COMUN DIVISOR ES =',A)
END.
```

Cuando el programa inicie su ejecución, la primera sentencia WRITE presentará en la pantalla el mensaje ENTRAR DOS NUMEROS ENTEROS y quedará a la espera de que el usuario escriba dos números mediante el teclado. Deberá escribirse: Primer número, espacio en blanco, Segundo número, tecla de retorno (Return). Los números deben separarse por al menos un espacio en blanco, no pueden separarse por comas. Los datos que se escriben mediante el teclado aparecen también en la pantalla (a este efecto se denomina eco).
A continuación el programa ejecutará el resto de sentencias, y finalmente la ultima sentencia WRITE presentará en la pantalla el mensaje EL MAXIMO COMUN DIVISOR ES = *valor resultante*, y finalizará la ejecución.

Las palabras INPUT y OUTPUT que aparecen en la cabecera del programa se denominan *parametros del programa* . INPUT ~~significa~~ indica que se utiliza el Teclado para la entrada de información en el programa, y OUTPUT que se utiliza la pantalla para la salida de información. Podrán existir también otros parámetros (ficheros de disco) para la entrada o salida de información.

El algoritmo anterior puede mejorarse si no se permite la entrada de números A y B que no sean positivos. En dicho caso, la entrada de estos valores respondería a la siguiente estructura repetitiva:



Con lo cual, la entrada de datos en el programa anterior quedaría codificada en Pascal con la siguiente sentencia Repeat:

```
REPEAT
  WRITE('ENTRAR DOS NUMEROS ENTEROS ');
  READ(A,B)
UNTIL  (A>0) AND (B>0);
```

Recuerdese la necesidad del uso de paréntesis debido a que el operador AND tiene una prioridad superior al operador relacional >. De esta forma, si se introducen dos datos A y B que no satisfacen la expresión booleana (A>0) AND (B>0), volverá a presentarse en pantalla el mensaje ENTRAR DOS NUMEROS ENTEROS, quedando el programa a la espera de que el usuario escriba, de nuevo, los dos valores solicitados. Cada vez que el programa sea ejecutado, podrán entrarse valores A y B cualesquiera (siempre que sean enteros positivos), obteniéndose su máximo común divisor.

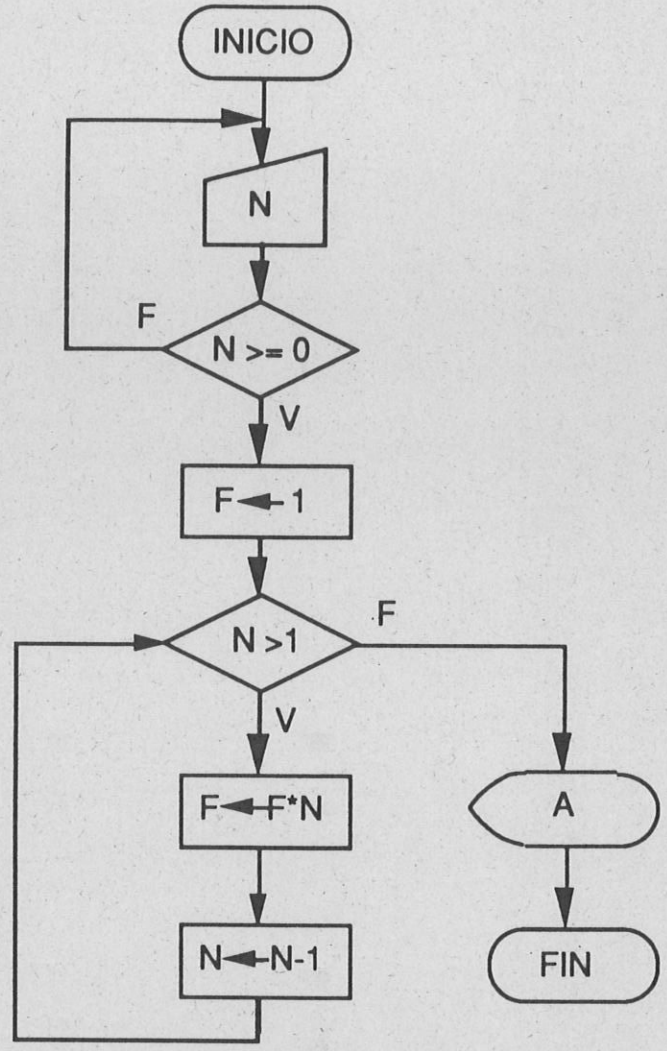
Ejemplo 2:
Factorial de un número:

Dado un valor N entero positivo, se desea diseñar el algoritmo y programa que permita obtener el factorial de N, que se define con:

Factorial de N = N! = N (N-1) (N-2)...3 .2
y por definición, 0! = 1

Por ejempo: 0! = 1
1! = 1
2! = 2
3! = 6
4! = 24

Emplearemos una variable entera F para el cálculo del factorial de N. A ésta variable daremos el valor inicial 1, y en cada etapa del proceso calcularemos un nuevo valor de F igual a su valor anterior multiplicado por N, decrementando este valor de N en 1 después de cada producto. El proceso finalizará cualdo el valor de N sea menor o igual a la unidad. El diagrama de este proceso será el siguiente:



El programa Pascal correspondiente al algoritmo anterior es:

```
PROGRAM FACTORIAL (INPUT, OUTPUT);
VAR   F, N : INTEGER;
BEGIN
REPEAT
   WRITE('N = ');
   READ(N)
UNTIL  N >= 0;
F := 1;
WHILE  N > 1  DO
BEGIN
   F := F*N;
   N := N-1
END;
WRITE('EL FACTORIAL ES = ', F)
END.
```

Puede observarse que para el caso N=0, se obtiene también el resultado correcto, al haber empleado la condición N>1, en lugar de N=1 (que permitiría la obtención de resultados válidos sólo para valores de N mayores que cero).

2. Traza de algoritmos o programas. Depuración.

En los algortimos y programas anteriores, por su sencillez, es posible observar con facilidad su correcto diseño para cualquiera de los posibles datos de entrada. Sin embargo, esta no suele ser la situación habitual. Los algoritmos pueden contener un elevado número de variables, y una gran diversidad de casos que puedan darse en los datos de entrada. Es por tanto aconsejable emplear algún tipo de análisis del algoritmo para corregir posible errores de diseño en el mismo, ya que ello nos ahorrará tiempo y esfuerzos inútiles al intentar compilar un programa que contenga estos errores de diseño.

Una herramienta útil para este fín es el estudio de la TRAZA del algoritmo para los diversos casos representativos de los datos que puedan presentarse.

Se define por TRAZA de un algoritmo (o de un programa) a la secuencia de valores que toman las variables del mismo desde el inicio hasta el final de proceso, en una serie de "puntos de observación" del proceso preestablecidos (en el instante anterior, o posterior a la realización de determinadas sentencias). La traza se referirá siempre a una elección particular de las informaciones primarias (datos) del proceso.
Definimos como Estado del algoritmo (referido a un punto de observación) al conjunto de valores de todas las varibles empleadas, en dicho punto.

Eligiendo de forma adecuada estos puntos, podremos verificar la correcta evolución de valores de las variables. Como máximo, tomaremos como puntos de observación todos los asociados a cada asignación de valor de alguna de las variables empleadas.

La traza se representará en forma de tabla, con tantas columnas como variables se utilicen, encabezadas por el nombre de la variable, y tantas filas como pasos se realicen en el proceso por algún "punto de observación". Cada fila corresonderá, pues, a un estado del algoritmo.

Las variables empleadas podrán pertenecer a uno o más de los siguientes grupos:

a) Variables que contienen valores de entrada al proceso, o valores iniciales prefijados.
b) Variables que contienen valores intermendios del proceso.
c) Varibables que contienen resultados finales del proceso.

Pueden no existir variables en alguno de estos grupos.

Una posible representeción de la traza de un algoritmo con 5 variables (V1 a V5) y 4 estados (e1 a e4) podría ser la siguiente. Se ha supuesto, para mayor simplicidad del ejemplo, que las variables son de tipo entero, y se ha tomado una evolución ficticia de valores.

Variables:	V1	V2	V3	V4	V5
Datos:	#	#			
e1	4	6	-	-	-
e2	4	6	-	12	-
e3	4	8	7	12	9
e4	4	8	9	15	9
Resultados:		#		#	

En este ejemplo, los datos (4, 6) se almacenan en las variables V1 y V2. Los resultados (8, 15) se obtienen en las variables V2 y V4. El resto de variables se utilizan para operaciones intermedias en el proceso. Observese que, en este ejemplo, la misma variable V2 contiene un dato, al inicio, y un resultado al final.
Puede facilitar el seguimiento del algoritmo, al utilizar la traza, añadir más columnas correspondientes a expresiones empleadas que no se asignan a ninguna variable. Este es, por ejemplo, el caso de expresiones utilizadas en los condicionales.

Veamos como ejemplo diversas trazas de los algoritmos diseñados en el apartado anterior:

1. Maximo común divisor de dos números: Traza.

Tomaremos 3 puntos de observación:
P1: Después de la entrada de A y B
p2: Después de la asignación de valor a la variable A
p3: Después de la asignación de valor a la variable B

Observaremos las siguientes expresiones booleanas:
A=B
A>B

Caso 1:

Datos: A = 28
B = 20

		Variables:		Expresiones:	
TRAZA:		A	B	A=B	A>B
		- -	- -	- - - - - -	- - - - - -
e1	P1	28	20	FALSE	TRUE
e2	P3	8	20	FALSE	FALSE
e3	P2	8	12	FALSE	FALSE
e4	P2	8	4	FALSE	TRUE
e5	P3	4	4	TRUE	FALSE

Resultado: MAXIMO COMUN DIVISOR ES 4

Caso 2:

Datos: A = 4
B = 3

		Variables:		Expresiones:	
TRAZA:		A	B	A=B	A>B
		- -	- -	- - - - - -	- - - - - -
e1	P1	5	3	FALSE	TRUE
e2	P3	2	3	FALSE	FALSE
e3	P2	2	1	FALSE	TRUE
e4	P2	1	1	TRUE	FALSE

Resultado: MAXIMO COMUN DIVISOR ES 1

2. Factorial de un número. Traza.

Tomaremos 2 puntos de observación:
P1: Despúes de la asignación F:=1
P2: Después de la asignación N:=N-1

Observaremos la siguiente expresión booleana:
N<=1

Caso 1:

Dato: N = 4

		Variables:		Expresiones:
TRAZA:		F	N	N<=1
		- -	- -	- - - - - -
e1	P1	1	4	FALSE
e2	P2	4	3	FALSE
e3	P2	12	2	FALSE
e4	P2	24	1	TRUE

Resultado: EL FACTORIAL ES 24

Caso 2:

Dato: N = 0

		Variables:		Expresiones:
TRAZA:		F	N	N<=1
		- -	- -	- - - - - -
e1	P1	1	0	TRUE

Resultado: EL FACTORIAL ES 1